|  |  |
| --- | --- |
| Ejercicios de repaso de Programación del CFGS DAM | Alejandro Micó González |

Contenido

[Ejercicio 1. LINAJE DE ROBOTS 1](#_Toc128859537)

[Ejercicio 2. YUM SNACK BAR 5](#_Toc128859538)

[Ejercicio 3. BIBLIOTECA FILIPINAS 9](#_Toc128859539)

# Ejercicio 1. LINAJE DE ROBOTS

|  |
| --- |
| Conocimientos necesarios: |
| Creación de un programa de consola en Java uso de Eclipse u otro IDE Programación Orientada a Objetos Estructuras básicas (variables, métodos, constantes, operadores) Control de flujo (condicionales, bucles) Visibilidad encapsulamiento Arrays, arrays bidimensionales Tipos enumerados (Enum) Sobreescritura de métodos Sobrecarga de métodos / sobrecarga de constructores Herencia y polimorfismo Uso de static y final *Casting* o refundición de tipos Uso de funciones aleatorias |

Escribe un programa en el lenguaje Java, utilizando Eclipse, consistente en una serie de clases de tipos de Robot destinados a realizar cálculos y operaciones en salida por consola como se describe en los siguientes puntos:

1. Define una superclase abstracta Robot que servirá de base para la creación de todos los robots. Un Robot contará siempre con un nombre y numeroSerie, ambos de tipo String. Crea los campos necesarios para la clase Robot y un constructor parametrizado apropiado.
2. La superclase abstracta Robot contará además con un campo estático contadorRobots que servirá para contar cuántos objetos de tipo Robot se han construido en total. Asimismo, define métodos *getters* y *setters* adecuados para los campos de la clase, excepto un setter para contadorRobots, que no debe ser definido pues se quiere evitar la falsificación externa del contador de construcción de robots.
3. Añade a la clase Robot, por último, una sobreescritura del método toString() propio de todo objeto en Java que devuelva la siguiente información “Soy un Robot de: [nombre=" + nombre + ", número de serie=" + numeroSerie.
4. Crea dos clases abstractas que hereden de Robot, RobotArtista y RobotCientifico. Ambas clases implementarán su propia sobreescritura del método toString() añadiendo a lo que devuelve el método toString() de su superclase, según corresponda, la información: “, disciplina=Ciencias]” o “, disciplina=Artes]”. Considera la posibilidad de que implementen su propio constructor coherente con el constructor de la clase Robot.
5. La clase RobotCientifico declarará un método abstracto procesar(double numero) para implementación y uso por las clases que hereden de ella. La clase RobotArtista definirá dos constantes “REGULAR” y “ALEATORIO” para definir el Comportamiento de las clases que hereden de ella ((considera la posibilidad de declararlas en un tipo enumerado de Java o Enum denominado Comportamiento). RobotArtista, por esta razón, debe contar con dos constructores sobrecargados, uno que reciba meramente por parámetros el nombre y numeroSerie y asigne automáticamente el Comportamiento “REGULAR” al RobotArtista así construido, y un constructor que especifique además de esos dos parámetros un tercer parámetro de Comportamiento, asignando al campo correspondiente el comportamiento que haya recibido como argumento.
6. Debes definir dos subtipos de RobotCientifico, RobotCortador y RobotPrimo. Estas dos subclases no podrán ser objeto de herencia por otras clases por debajo de ellas.
7. La clase RobotPrimo ofrece dos utilidades. 1) Su implementación o concreción de procesar(double numero) evalúa si el número pasado por argumento es o no un número primo y devuelve un String con un mensaje que lo indica. Son números primos aquellos números naturales que sólo son divisibles por sí mismos y por 1. Considera la introducción de lógica de control que evalúe si el número recibido por parámetro es negativo o tiene decimales, casos en los que un número no puede ser primo. 2) La segunda utilidad de RobotPrimo será un método para imprimir todos los números primos que existen en un intervalo, por esta razón este método –dale un nombre apropiado— debe recibir por parámetro dos números enteros, el principio y el fin de dicho intervalo.
8. La clase RobotCortador implementa un catálogo de métodos sobrecargados procesar(). El que sobreescribe el método abstracto definido por RobotCientifico y recibe un valor double por parámetro, imprime en consola la parte entera y la parte decimal por separado. La clase además definirá: un método procesar(int numero) que simplemente devuelve el valor recibido dividido por 2 y un método procesar(String string) que divide la cadena recibida en dos mitades y las imprime por separado en consola. Considera opcionalmente crear dos métodos procesar() más que reciben por parámetro un array de enteros y otro un array de Strings, los dividen por la mitad e imprimen los arrays resultantes por consola.
9. Debes definir dos subtipos de RobotArtista, RobotPoeta y RobotPintor. Estas dos subclases no podrán ser objeto de herencia por otras clases por debajo de ellas.
10. RobotPintor tiene dos constructores alineados con los de su superclase RobotArtista: uno que recibe solo nombre y numeroSerie y otro que, además, recibe un Comportamiento. Esta clase contiene definiciones constantes y estáticas de tres figuras geométricas, CUADRADO, TRIÁNGULO y ROMBO (pueden incluirse en un tipo Enum Figura; un método dibujar(Figura figura, int altura, char caracter) que dibuja en consola cualquiera de las tres figuras, compuesta de una repetición de caracteres del valor char pasado por argumento y de una altura en líneas de consola según la altura pasada por argumento. Ejemplos:

|  |  |  |  |
| --- | --- | --- | --- |
| m  mmm  m | $  $$  $$$  $$$$  $$$$$ | u u  u u | p p p p p p  p p p p p p  p p p p p p  p p p p p p  p p p p p p  p p p p p p |

1. Considera el uso de una estructura de tipo switch-case en el método de RobotPintor dibujar() para determinar el dibujo que se realizará, y la posibilidad de delegar el trabajo de los bucles de impresión en consola de tales dibujos en unos métodos dibujarCuadrado(), dibujarTriangulo(), dibujarRombo();
2. Los objetos de RobotPintor instanciados con un Comportamiento REGULAR dibujarán exactamente la figura con la altura y carácter de repetición pasados por argumento, mientras que los RobotPintor de Comportamiento ALEATORIO ignorarán los parámetros de configuración de dibujar() y pintarán una figura aleatoriamente elegida, de altura aleatoria (máximo 12 líneas) y un carácter aleatorio (considera la posibilidad de restringir los caracteres a un rango concreto de caracteres de la tabla ASCII, p. ej. entre 33-95 para evitar la impresión de figuras “invisibles”).
3. La clase RobotPoeta contiene tres campos de tipo String[] (array de Strings), cada uno de ellos con 4 líneas de un poema o canción diferentes y un campo String[][] (array de arrays) que almacena los tres poemas. La clase definirá dos constructores sobrecargados alineados con los de su superclase RobotArtista y un único método recitar(), que no devuelve nada pero imprime en consola 4 versos de la colección bajo las siguientes reglas: Si RobotPoeta tiene un Comportamiento REGULAR, imprimirá los 4 versos del poema o canción de uno de los arrays String[] elegido al azar de manera congruente.; si tiene un Comportamiento ALEATORIO, imprimirá 4 versos escogidos aleatoriamente de cualquiera de los tres poemas o canciones, de modo que existirá una probabilidad muy pequeña de que imprima 4 versos del mismo poema en su debido orden.
4. Define, por último, una clase principal con método main() destinada a instanciar un Robot de cada tipo (un RobotPrimo, un RobotCortador, un RobotPintor REGULAR, un RobotPintor ALEATORIO, un RobotPoeta REGULAR y un RobotPoeta ALEATORIO), imprimir la información general de cada objeto mediante el método toString() e invocar a continuación los métodos de cada objeto para someterlos a prueba en la ejecución del programa. El orden de pruebas propuesto es el siguiente:
   1. Impresión de datos de los seis objetos.
   2. Solicitar al RobotPrimo procesar el número 11 para saber si es o no primo.
   3. Solicitar al RobotPrimo imprimir el intervalo de números primos entre 1 y 50.
   4. Solicitar a un RobotPintor de Comportamiento REGULAR dibujar un TRIÁNGULO de 8 líneas de alto compuesto de caracteres ‘$’.
   5. Solicitar a un RobotPintor de Comportamiento ALEATORIO imprimir una figura con cualesquiera argumentos: el objeto dibujará una figura de manera aleatoria.
   6. Solicitar al RobotCortador procesar el número 4.0 y procesar el número 0.285714.
   7. Solicitar al RobotCortador procesar el String “acurruca”, procesar el array de enteros {5, 5, 45, 66, 1001, 2, 3, 11} y procesar también el array de Strings {"Mercurio", "Venus", "Tierra", "Marte", "Júpiter", "Saturno", "Urano", "Neptuno"}.
   8. Solicitar a un RobotPoeta de Comportamiento REGULAR recitar versos de su colección.
   9. Solicitar a un RobotPoeta de Comportamiento ALEATORIO recitar versos de su colección.
   10. Imprime para terminar un mensaje indicando el número total de Robots construidos.

Se ofrece a continuación un ejemplo de salida por consola del programa completo:

I am a Robot of: [name=Bender, serialNumber=1010011010BU22, discipline=Science]

I am a Robot of: [name=Euclid, serialNumber=E213002248G, discipline=Science]

I am a Robot of: [name=Frida, serialNumber=K15019071954L, discipline=Arts]

I am a Robot of: [name=Salvador, serialNumber=D1904858585A, discipline=Arts]

I am a Robot of: [name=Byron, serialNumber=E18091849AP, discipline=Arts]

I am a Robot of: [name=3DG4R-4LL4N-P03, serialNumber=E18091849AP, discipline=Arts]

+----------------------+

| Operations |

+----------------------+

11 is a prime number.

Primes between 1-50:

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

$

$$

$$$

$$$$

$$$$$

$$$$$$

$$$$$$$

$$$$$$$$

G

GGG

G

Whole part: 4; Decimal part: 0.0

Whole part: 0; Decimal part: 0.285714

acur / ruca

First half: [5, 5, 45, 66] / Second half: [1001, 2, 3, 11]

First half: [Mercurio, Venus, Tierra, Marte] / Second half: [Júpiter, Saturno, Urano, Neptuno]

3-1 Tu rencor, tu porqué, tu agonía...

3-2 O tal vez esa sombra

3-3 que se tumba a tu lado en la alfombra

3-4 A esperar que suba la marea.

1-4 Y no poder del árbol desasirse

2-1 ¿Qué es poesía?, dices mientras clavas

3-1 Tu rencor, tu porqué, tu agonía...

2-2 en mi pupila tu pupila azul.

=====Total robots manufactured: [6]=====

# Ejercicio 2. YUM SNACK BAR

|  |
| --- |
| Conocimientos necesarios: |
| Programación Orientada a Objetos Composición Modularización en paquetes e importaciones Uso de Colecciones dinámicas en Java Creación y captura de excepciones Creación e implementación de Interfaces API de tiempo/fechas de Java 8+ Persistencia de información, Java I/O de ficheros Comparación de objetos (*Comparable*) |

El objetivo es crear un programa en Java para administrar los aspectos básicos de la venta de productos de un establecimiento de palomitas, bebidas y aperitivos para consumir en el cine. La función principal es el registro de las compras o transacciones, simulando una aplicación de TPV o "cash register".

El establecimiento "Yum Snack Bar" ofrece a sus clientes los siguientes productos: 1) Refrescos y 2) palomitas de diferentes tamaños; 3) chocolatinas o snacks, 4) bolsas de chucherías al peso y 5) bolsas de frutos secos variados que se compran también al peso o granel.

1. Construye una solución Orientada a Objetos que contemple las diferentes clases necesarias para organizar el programa, utilizando paquetería para dividir lo mejor posible el código agrupando las clases por sus características y finalidad.
2. Parece clara la conveniencia de utilizar la herencia: por ejemplo, es posible concebir una clase CajaPalomitas y otra clase VasoRefresco, con características propias, y una superclase Producto con características comunes como un nombre y un precio.
3. Crea unas constantes que definan los tamaños posibles de las palomitas y las bebidas: "MEDIANO", "GRANDE", "GIGANTE". Una solución posible es la definición de un tipo enumerado Tamano.
4. Las diferentes clases de productos deben implementar los métodos toString() y *getters* y *setters* necesarios. Los constructores de clases y superclases deben coordinarse según los requisitos del ejercicio. Ten presente que no se pasará el nombre ni el precio de los productos en su instanciación, sino que tendrán un nombre descriptivo y precios yapredefinidos. Parece aconsejable utilizar a tal fin campos estáticos, p. ej. (double) PRECIO, a falta del uso de una base de datos de productos .
5. Las palomitas y bebidas deben tener diferentes precios en función del tamaño. El constructor de la clase VasoRefresco, además del Tamano, recibirá por parámetro el tipo (“COLA”, “COLA\_LIGHT”, “NARANJA”, “LIMON”) para lo cual se recomienda el uso de un tipo Enum anidado en la clase. Existirá una clase Menu que permitirá la compra de la combinación de productos palomitas+bebida a precios especiales de promoción. Es aconsejable utilizar la composición para diseñar esta clase, que constará de objetos CajaPalomitas y VasoRefresco.
6. El constructor de Menu evaluará que las palomitas y bebidas pasadas por argumento sean del mismo tamaño. No se permiten menús con un caja de palomitas y un vaso de bebida de diferente tamaño (p.ej. Palomitas MEDIANO y Refresco GRANDE). Crea una excepción personalizada llamada 'ExcepcionTamanoIlegal' que sea arrojada en tales casos.
7. Las clases para BolsaChucherias y BolsaFrutosSecos, que permiten la compra al peso, deben fijar un precio estático por 100g y su constructor recibirá por parámetro el peso concreto de la compra, que se multiplicará por aquel atributo para calcular el precio. Crea una Interfaz denominada Pesable con un método calcularPrecio(double peso) para sea implementado por estas dos clases de productos que se venden a granel. Es aconsejable emplear alguna función de redondeo de 2 decimales para evitar la impresión de precios como 8.250000000000002.
8. Las clases BolsaChucherias y BolsaFrutosSecos deben incorporar una lógica que arroje una excepción IllegalArgumentException si se intenta comprar una bolsa con un precio negativo o anormalmente bajo, menor de 20 g, para prevenir trampas con la balanza.
9. La clase Chocolatina será la más simple de todas: hereda de Producto, tendrá un precio estático, el nombre estático del producto y un constructor que no recibe argumentos.
10. Crea una clase que Tienda almacene la información del establecimiento que consideres necesaria, y como mínimo su nombre y la fecha en que la tienda fue fundada. Utiliza preferiblemente alguna de las clases relativas a fechas de la API de Java 8 o superior. Debes crear un método que permita imprimir en consola un mensaje del tipo: "Bienvenido al programa de Yum Snack Bar, sirviendo palomitas, colesterol y alegría desde hace {16} años". Este número de años debe calcularse de manera dinámica en función de la fecha de fundación predefinida de la tienda y su diferencia con la fecha actual del sistema cuando se ejecute el programa.
11. Crea una clase Transaccion destinada a aglutinar todos los productos comprados por un solo cliente en una operación de caja (p. ej., un cliente compra un Menu, una chocolatina y una bolsa de chucherías en una sola transacción). Una Transaccion se compone de un entero id, una fechaHora exactas de su creación, una lista de <Producto>s (utiliza una colección dinámica) y el precioTotal de los productos que la conforman. Crea los métodos necesarios en la clase Transaccion, incluyendo toString() para mostrar sus componentes en consola, y utiliza los métodos provistos por la clase Producto para implementar la lógica.
12. Se necesita conocer cuál es la Transaccion más cara (precioTotal más elevado) de una lista de transacciones. Puede conseguirse este objetivo de diversas maneras, pero se recomiendo utilizar la interfaz Comparable o la interfaz Comparator para conseguirlo.
13. Crea una clase LogTransaccion destinada a almacenar en un fichero 'log.txt' la información de todas las transacciones de una sesión del programa y capaz también de leer la información del log y mostrarla por consola. El formato del log y los detalles de implementación son libres, pero se recomienda guardar cada transacción en una línea, con un formato p. ej.: 'fecha-hora trans-id, producto, producto, producto: precio'.
14. Como ejercicio adicional, codifica la clase LogTransaccion con la lógica de comprobación necesaria para que la transacción más cara (punto 12) se guarde en el fichero añadiendo al final de la línea el string “(!)”.
15. En una clase principal, en el método main(), invoca la frase de presentación de la tienda (punto 8) y crea sucesivas ventas de productos y transacciones para someter el programa a prueba; las operaciones a desarrollar en el método principal son las siguientes:
    1. Se crearán secuencialmente seis transacciones diferentes (Transaccion) con el fin de guardarlas en una lista, pasarlas a LogTransaccion para guardarlas en el fichero de log y a continuación leer ese fichero e imprimir su contenido en consola.
    2. Para la primera transacción, crea las compras de una CajaPalomitas “GRANDE” y una chocolatina, guarda la transacción en la lista de transacciones e imprímela.
    3. Para la segunda transacción, compra un VasoRefresco “MEDIANO” y una BolsaFrutosSecos de 400g, guarda la transacción e imprímela.
    4. Para la tercera transacción, compra un menú compuesto de un VasoRefresco de “COLA\_LIGHT” “GIGANTE” y una CajaPalomitas “MEDIANO”, forzando el lanzamiento (y captura) de la ExcepcionTamanoIlegal (punto 6). La excepción debe prevenir que esta **transacción inválida** se guarde, loguee o imprima en consola.
    5. Para la cuarta transacción, compra una BolsaChucherias haciendo trampas con la balanza, por un peso de 6 g, forzando el lanzamiento de la IllegalArgumentException del punto 8. La excepción debe prevenir que esta **transacción inválida** se guarde, loguee o imprima en consola.
    6. Para la quinta transacción, compra una BolsaChucherias de 400g y un Menu compuesto de un VasoRefresco “NARANJA” “GIGANTE” y una CajaPalomitas “GIGANTE”. Intenta instanciar el objeto Transaccion, los objetos Producto y los objetos que componen el Menu en una sola instrucción de código (instanciación de objetos anónimos o instanciación “al vuelo”). Guarda la transacción e imprímela.
    7. Para la sexta transacción, que será la más cara, compra una BolsaChucherias de 750g y un VasoRefresco de “COLA” “GIGANTE”, guarda la transacción e imprímela.
    8. Imprime en consola cuál es la transacción más cara de toda la sesión.
    9. Por último, guarda la lista de transacciones en el fichero de log y recupera la información que se ha guardado en este fichero para imprimirla por consola.

|  |  |
| --- | --- |
| Sugerencia sobre el precio de los productos | |
| Palomitas MEDIANO / GRANDE / GIGANTE | 5.0 / 7.0 / 9.0 |
| Refresco MEDIANO / GRANDE / GIGANTE | 4.5 / 6.0 / 7.5 |
| Menu MEDIANO / GRANDE / GIGANTE | 7.95 / 8.95 / 9.95 |
| Chocolatina | 2.0 |
| Bolsa de frutos secos (100 g) | 1.80 |
| Bolsa de chucherías (100 g) | 1.10 |

A continuación se expone el ejemplo de una salida por consola del programa completo:

+------------------------+

| Welcome to the Yum Snack Bar|

| Puerta del Sol, Km. 0, 28013 Madrid, Spain|

| Serving popcorn, cholesterol, and joy for 37 years. |

+------------------------+

Transaction [ID=1,

formattedTimestamp=15/02/2023-15:00:33,

items=[Popcorn box 'LARGE' - price=7.0], Product: 'Chocolate snack' - price=2.0],

totalCost=9.0]

Transaction [ID=2,

formattedTimestamp=15/02/2023-15:00:33,

items=[Soda beverage cup 'MEDIUM' - price=4.5], Product: 'Mixed nuts bag' - price=7.2],

totalCost=11.7]

products.IllegalSizeException: Soda cup and popcorn box must be of the same size.

at products.Menu.<init>(Menu.java:16)

at store.Main.main(Main.java:42)

java.lang.IllegalArgumentException: CandyBag weight must be 20g or higher

at products.CandyBag.<init>(CandyBag.java:18)

at store.Main.main(Main.java:55)

Transaction [ID=5,

formattedTimestamp=15/02/2023-15:00:33,

items=[Product: 'Gummy candy bag' - price=4.4, Menu: Soda+Popcorn 'EXTRALARGE' - price=9.95],

totalCost=14.35]

Transaction [ID=6,

formattedTimestamp=15/02/2023-15:00:33,

items=[Product: 'Gummy candy bag' - price=8.25, Soda beverage cup 'EXTRALARGE' - price=7.5]],

totalCost=15.75]

The most expensive transaction (!) is:

Transaction [ID=6,

formattedTimestamp=15/02/2023-15:00:33,

items=[Product: 'Gummy candy bag' - price=8.25, Soda beverage cup 'EXTRALARGE' - price=7.5]],

totalCost=15.75]

==== Transactions in the log file ====

15/02/2023-15:00:33 ID=1, Popcorn box, Chocolate snack; price=9.0

15/02/2023-15:00:33 ID=2, Soda beverage cup, Mixed nuts bag; price=11.7

15/02/2023-15:00:33 ID=5, Gummy candy bag, Soda cup + Popcorn box at menu price; price=14.35

15/02/2023-15:00:33 ID=6, Gummy candy bag, Soda beverage cup; price=15.75 (!)

# Ejercicio 3. BIBLIOTECA FILIPINAS

|  |
| --- |
| Conocimientos necesarios: |
| Programación Orientada a Objetos en Java (composición y otras técnicas) Colecciones en Java Clases e interfaces de la API JDBC de Java Bases de datos relacionales: conceptos y estructura Lenguaje MySQL (DDL, DML, DQL) Instalación de un servidor MySQL local (p.ej. phpMyAdmin) Descarga de un driver JDBC apropiado e instalación en el classpath del proyecto Mapeo objeto-relacional manual (sin el uso de un framework ORM) Sintaxis CRUD básica en MySQL Uso de Prepared Statements Cierre de recursos para evitar fugas de memoria (opción de empleo de la sintaxis try-with-resources) Captura de excepciones Buenas prácticas en la escritura de código |

El objetivo de este ejercicio es crear un programa en Java con utilidad práctica capaz de administrar el servicio de préstamo de libros de una biblioteca de uso público. La aplicación realiza consultas y actualizaciones de una base de datos MySQL cuyas sentencias de creación e inserción de datos se proporcionan al final de este enunciado.

La base de datos almacena información en cuatro tablas: `library` con información y reglas generales de la biblioteca; `book` con información de los libros en los fondos de la biblioteca y su disponibilidad; `reader` con información sobre los lectores o usuarios abonados al servicio de préstamo de la biblioteca; y `borrowing` con información sobre los préstamos de libros a los lectores, que incluye claves foráneas que hacen referencia a las tablas `book` y `reader`.

Es importante conocer las **Normas de Préstamo** de la biblioteca para desarrollar el software de administración de manera congruente\*. La biblioteca diseñada por defecto en las sentencias SQL que se ofrecen más abajo: 1) Permite a cada usuario tener en préstamo un máximo de 3 libros al mismo tiempo; 2) Concede los préstamos por un plazo máximo de 15 días; 3) Y por último, como es típico en muchas bibliotecas públicas de España, fija una penalización para los lectores que devuelvan libros más tarde del plazo máximo, en este caso una penalización de 3 días por cada día de retraso en la devolución de un libro.

Conociendo estos requisitos, realiza un programa en Java con las siguientes especificaciones:

1. Crea un proyecto que incorpore en el classpath el driver necesario para manejar una **base de datos MySQL** y define en el programa en Java la ruta, credenciales y otros valores necesarios para establecer la conexión con un servidor local.
2. Crea la base de datos e inserta los valores que se proporcionan al final de este enunciado en un **servidor local** (localhost) MySQL (puedes utilizar p.ej. XAMPP-phpMyAdmin o MySQL Workbench). Los registros de ejemplo suministrados ofrecen una variedad de casos para poder someter el programa que desarrolles a prueba, incluyendo préstamos (“borrowings”) dinámicamente creados en fechas relativas a la fecha del sistema en el momento de inserción de los datos.
3. Crea las clases necesarias en tu proyecto en Java para ofrecer una **solución orientada a objetos** con la suficiente modularidad. Esto significa que el programa debe tener clases “modelo” equivalentes a las entidades de la base de datos: Library, Book, Reader y Borrowing con los campos y miembros que corresponden a los campos de la tabla. Se recomienda dividir el código en clases y paquetes siguiendo un patrón de diseño **MVC** o arquitecturas modulares similares.
4. El software de administración de una biblioteca como la presentada en este enunciado podría ofrecer muchas funcionalidades tales como operaciones CRUD de libros y lectores abonados, devolución de listas, búsquedas y comparaciones con filtros avanzados, etc. Sin embargo, en este ejercicio sólo debes **programar la funcionalidad necesaria para realizar préstamos de libros a lectores y registrar devoluciones de los libros prestados.** A pesar de que estas dos funciones parecen sencillas, implementarlas adecuadamente con arreglo a las reglas descritas al principio exigirá realizar consultas y comprobaciones en las cuatro tablas.
5. El programa solicitará al usuario con un “menú” por **consola**, en un **bucle infinito**, qué opción quiere realizar —PRESTAMO, DEVOLUCION, SALIR— y a continuación solicitará los valores de ID del libro e ID del lector para llevar a cabo la operación seleccionada, si es posible realizarla, mediante métodos especializados.
6. Debes poner especial atención en que el programa, para realizar préstamos y devoluciones, además de otras situaciones evalúe las reglas de número máximo de préstamos, plazo máximo de días de préstamo y días de sanción por día de retraso en el préstamo de la biblioteca, así como el número de préstamos que tiene un lector, si tiene o no días de sanción, el valor de disponibilidad (“available”) de un libro para saber si está prestado actualmente o no, etcétera.
7. Debe controlarse un caso especial que consiste en el intento de devolver un libro actualmente prestado a un usuario por otro usuario diferente. Si se permitiera hacerlo, aunque el cambio en la disponibilidad del libro se registraría coherentemente, el sistema permitiría el caso indeseado de que un lector viera reducido su número de libros prestados incluso a valores negativos, alterando las normas de funcionamiento de la biblioteca. Para evitarlo, el método que maneja las devoluciones debe comprobar que hay un préstamo del libro especificado al lector introducido y que este préstamo está vigente (la fecha de devolución es nula).
8. **OPCIONAL:** Puedes intentar implementar la lógica más compleja de este programa: haz que evalúe los días que un lector lleva de retraso en la devolución de libros que todavía tiene prestados y: 1) Impida tomar prestados nuevos libros a ese lector si tiene retraso; 2) Aplique la penalización que le corresponde sumándola al valor del campo `penalty\_count` de su registro en la tabla `reader` y al campo `overdue\_penalty` del registro de ese préstamo en la tabla `borrowing` al tiempo de la devolución.
9. OPCIONAL: Crea una interfaz gráfica con Java Swing u otro framework similar para facilitar la gestión de préstamos y devoluciones de libros en una interfaz de usuario.
10. OPCIONAL: Añade otras funciones que consideres interesantes para la aplicación.
11. **RECOMENDACIÓN:** Para buscar en la base de datos préstamos por su ID en orden a registrar una DEVOLUCIÓN, se recomienda limitar la *query* a la base de datos al último resultado (cláusulas DESC LIMIT 1) para evitar confusiones con préstamos antiguos del mismo libro al mismo lector que podrían producir un funcionamiento anómalo del programa.
12. Para someter a prueba el programa, se realizará una ejecución dando las siguientes órdenes en el menú de selección de opciones:
    1. La primera operación consistirá en registrar un PRESTAMO del libro con ID 1 (“El Quijote”) por el lector con ID 1 (George Stobbart). Se mostrará un mensaje de confirmación, se modificará la disponibilidad del libro y se incrementará el contador de libros prestados de este lector.
    2. La segunda operación consistirá en practicar la DEVOLUCION del libro con ID 3 (“Las aventuras de Huckleberry Finn”) prestado al lector con ID 1 (George Stobbart). Este préstamo viene preconstituido por las inserciones propuestas al final del enunciado, diseñadas para que se realice con fecha del día mismo de la inserción a las 08:34:05 horas. Se mostrará una confirmación de la devolución, se cambiará la disponibilidad del libro (true) , se insertará la fecha de devolución en el registro correspondiente de la tabla `borrowing` y se disminuirá el contador de libros prestados del lector.
    3. La tercera operación consistirá en intentar registrar un PRESTAMO del libro con ID 1 (“El Quijote”) por el lector con ID 6 (Liam McGuire), debiendo mostrarse el mensaje de que no es posible realizar el préstamo tras comprobarse el estado no disponible del libro a causa de la primera operación.
    4. La cuarta operación consistirá en intentar registrar un PRESTAMO del libro con ID 10 (“Vida y destino”) por el lector con ID 4 (Augustin Rosso), debiendo mostrarse el mensaje de que no es posible autorizar el préstamo porque el lector ya tiene 3 libros, el número máximo de préstamos permitidos.
    5. La quinta operación consistirá en intentar registrar un PRESTAMO del libro con ID 14 (“Los miserables”) por el lector con ID 15 (Gamal Khan), debiendo mostrarse el mensaje, a pesar de que el libro está disponible, de que no es posible autorizar el préstamo porque el lector tiene una penalización por retrasos vigente.
    6. La sexta operación consistirá en intentar registrar la DEVOLUCIÓN del libro con ID 11 (“La invención de Morel”) por la lectora con ID 9 (Pearl Henderson). Este libro en realidad está prestado al lector con ID 4 (Augustin Rosso) y por lo tanto, el sistema no autorizará la operación informando de esta circunstancia.
    7. La séptima operación consistirá en registrar la DEVOLUCION del libro con ID 9 (“El perro de los Baskerville”) por el lector con ID 12 (François Plantard), debiendo autorizarse la operación, mostrarse confirmación y practicarse las debidas operaciones de actualización de la base de datos en cuanto al número de libros prestados del lector, la fecha de devolución del préstamo y la disponibilidad del libro devuelto.
    8. **OPCIONAL:** la octava operación, si se ha implementado la lógica de control del **punto 8** opcional, consistirá en intentar registrar el PRESTAMO del libro con ID 9 (“El perro de los Baskerville”) devuelto por otro lector por la lectora con ID 2 (Nicole Collard). Esta lectora no presenta todavía una sanción registrada, pero tiene en préstamo dos libro desde hace 19 y 17 días respectivamente conforme a los datos preconstituidos al final del enunciado, por tanto se ha retrasado en su devolución y se espera que el programa prohíba el nuevo préstamo por este motivo.
    9. **OPCIONAL:** la novena operación, si se ha implementado la lógica de control del punto 8 opcional, consistirá en registrar la DEVOLUCIÓN del libro con ID 15 (“Las metamorfosis”) por el lector con ID 3 (André Lobineau), quien ya lleva 1 día de retraso en su devolución. La operación debe ser autorizada y se espera que el programa, además de modificar los campos de disponibilidad del libro, fecha de devolución del préstamo y número de libros prestados de este lector, actualice la base de datos incluyendo la penalización de 3 días por 1 día de retraso.
    10. **OPCIONAL:** las operaciones décima y undécima, si se ha implementado la lógica de control del punto 8 opcional, consistirán en registrar la DEVOLUCIÓN de los libros con ID 7 y 12 (“Robinson Crusoe” y “The stand”) de la lectora con ID 2 (Nicole Collard). Al tener un retraso de 4 y 2 días respectivamente en su devolución, se espera que el programa las registre correctamente y actualice la base de datos poniendo a la lectora Nicole Collard una penalización de 12 + 6 = 18 días.
    11. La última operación consistirá en elegir la opción SALIR, que deberá terminar con la ejecución iterativa y finalizar el programa. Conviene comprobar en la base de datos MySQL del servidor si los datos de las tablas `book`, `borrowing` y `reader` se han actualizado conforme a los préstamos y devoluciones tramitados durante el ejercicio.

A continuación se muestra la salida en consola producto de las pruebas propuestas:

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

borrow

Enter book ID: 1

Enter reader ID: 1

Book El Quijote borrowed successfully by George Stobbart and set 'unavailable'.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 3

Enter reader ID: 1

Book 'Las aventuras de Huckleberry Finn' returned successfully and set 'available'.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

borrow

Enter book ID: 1

Enter reader ID: 6

The book 'El Quijote' is already borrowed.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

borrow

Enter book ID: 10

Enter reader ID: 4

Reader Augustin Rosso has already the maximum borrowings.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

borrow

Enter book ID: 14

Enter reader ID: 15

Reader Gamal Khan has a late return or penalty and can't borrow books.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 11

Enter reader ID: 9

No borrowing found for the specified book and reader IDs.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 9

Enter reader ID: 12

Book 'El perro de los Baskerville' returned successfully and set 'available'.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

borrow

Enter book ID: 9

Enter reader ID: 2

Reader Nicole Collard has a late return or penalty and can't borrow books.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 15

Enter reader ID: 3

Book 'Las metamorfosis' returned successfully and set 'available'.

Late return penalty applied: 3 days.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 7

Enter reader ID: 2

Book 'Robinson Crusoe' returned successfully and set 'available'.

Late return penalty applied: 12 days.

+--------------------------------------------------------------+

| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

return

Enter book ID: 12

Enter reader ID: 2

Book 'The stand' returned successfully and set 'available'.

Late return penalty applied: 6 days.

+--------------------------------------------------------------+

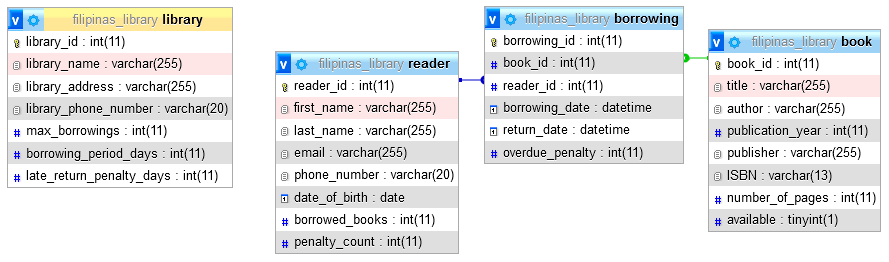
| Do you want to borrow or return a book? (Borrow/Return/Exit) |

+--------------------------------------------------------------+

exit

Exiting program...

|  |
| --- |
| Sentencias de creación de la base de datos **filipinas\_library** (DDL) |
| DROP DATABASE IF EXISTS filipinas\_library;  CREATE DATABASE filipinas\_library;  USE filipinas\_library;  CREATE TABLE library (  library\_id INT AUTO\_INCREMENT PRIMARY KEY,  library\_name VARCHAR(255),  library\_address VARCHAR(255),  library\_phone\_number VARCHAR(20),  max\_borrowings INT DEFAULT 3,  borrowing\_period\_days INT DEFAULT 15,  late\_return\_penalty\_days INT DEFAULT 3  );  CREATE TABLE book (  book\_id INT AUTO\_INCREMENT PRIMARY KEY,  title VARCHAR(255),  author VARCHAR(255),  publication\_year INT,  publisher VARCHAR(255),  ISBN VARCHAR(13),  number\_of\_pages INT,  available BOOLEAN DEFAULT TRUE  );  CREATE TABLE reader (  reader\_id INT AUTO\_INCREMENT PRIMARY KEY,  first\_name VARCHAR(255),  last\_name VARCHAR(255),  email VARCHAR(255),  phone\_number VARCHAR(20),  date\_of\_birth DATE,  borrowed\_books INT DEFAULT 0,  penalty\_count INT DEFAULT 0  );  CREATE TABLE borrowing (  borrowing\_id INT AUTO\_INCREMENT PRIMARY KEY,  book\_id INT,  reader\_id INT,  borrowing\_date DATETIME,  return\_date DATETIME,  overdue\_penalty INT DEFAULT 0,  FOREIGN KEY (book\_id) REFERENCES book(book\_id),  FOREIGN KEY (reader\_id) REFERENCES reader(reader\_id)  ); |
| Sentencias de inserción de registros en la base de datos **filipinas\_library** (DML) |
| INSERT INTO library (library\_name, library\_address, library\_phone\_number, max\_borrowings, borrowing\_period\_days, late\_return\_penalty\_days)  VALUES  ('Biblioteca Islas Filipinas', 'Calle Jesús Maestro, 3', '915800811', 3, 15, 3);  INSERT INTO book (title, author, publication\_year, publisher, ISBN, number\_of\_pages, available)  VALUES  ('El Quijote', 'Miguel de Cervantes', 1605, 'Francisco de Robles', '9788424921862', 863, true),  ('Moby Dick', 'Herman Melville', 1851, 'Harper & Brothers', '9781593080275', 635, false),  ('Las aventuras de Huckleberry Finn', 'Mark Twain', 1884, 'Chatto & Windus', '9780486280615', 224, false),  ('Fundación', 'Isaac Asimov', 1951, 'Gnome Press', '9780553293357', 244, false),  ('El Señor de los Anillos', 'J.R.R. Tolkien', 1954, 'George Allen & Unwin', '9780618260584', 1178, true),  ('El doctor Jekyll y Mr. Hyde', 'Robert L. Stevenson', 1886, 'Longmans, Green, and Co.', '9780141389509', 144, false),  ('Robinson Crusoe', 'Daniel Defoe', 1719, 'W. Taylor', '9780060771504', 320, false),  ('No es país para viejos', 'Cormac McCarthy', 2005, 'Vintage', '9780307387134', 320, true),  ('El perro de los Baskerville', 'Arthur Conan Doyle', 1902, 'George Newnes Ltd', '9780143122065', 256, false),  ('Vida y destino', 'Vasily Grossman', 1980, 'Ediciones Destino', '9788432209469', 896, true),  ('La invención de Morel', 'Adolfo Bioy Casares', 1940, 'Editorial Losada', '9789500301429', 144, false),  ('The stand', 'Stephen King', 1978, 'Doubleday', '9780385121682', 823, false),  ('Rebelión en la granja', 'George Orwell', 1945, 'Secker and Warburg', '9780451526342', 112, true),  ('Los miserables', 'Victor Hugo', 1862, 'A. Lacroix, Verboeckhoven & Cie', '9780192839983', 1232, true),  ('Las metamorfosis', 'Ovidio', 8, 'Varios', '9780140447897', 723, false);    INSERT INTO reader (first\_name, last\_name, email, phone\_number, date\_of\_birth, borrowed\_books, penalty\_count)  VALUES  ('George', 'Stobbart', 'george.stobbart@gmail.com', '626461866', '1979-06-18', 1, 0),  ('Nicole', 'Collard', 'nico.collard@yahoo.com', '612445112', '1984-10-03', 2, 0),  ('André', 'Lobineau', 'andre.lobineau@gmail.com', '691557349', '1985-12-29', 1, 0),  ('Augustin', 'Rosso', 'augustin.rosso@yahoo.com', '680220093', '1971-04-05', 3, 0),  ('Pierre', 'Carchon', 'pierre.carchon@bbc.co.uk', '616596154', '1999-01-12', 0, 0),  ('Liam', 'McGuire', 'liam.mcguire@facebook.com', '677916387', '2002-07-30', 0, 0),  ('Clarissa', 'Piermont', 'clarissa.piermont@altavista.com', '648915477', '1956-09-02', 0, 0),  ('Duane', 'Henderson', 'duane.penderson@hotmail.com', '610468251', '1982-03-21', 0, 0),  ('Pearl', 'Henderson', 'pearl.henderson@hotmail.com', '691465003', '1983-12-01', 1, 0),  ('Bruno', 'Ostvalt', 'bruno.ostvalt@yahoo.com', '611025916', '1965-08-08', 0, 0),  ('Nigel', 'Peagram', 'nigel.peagram@lycos.com', '696163171', '1991-02-17', 0, 0),  ('François', 'Plantard', 'francois.plantard@google.com', '625526744', '1970-01-10', 1, 5),  ('Jacques', 'Marquet', 'jacques.marquet@yahoo.com', '682412110', '1968-12-31', 0, 0),  ('Antoine', 'Eklund', 'antoine.eklund@gmail.com', '646826952', '1975-06-10', 0, 0),  ('Gamal', 'Khan', 'khan@google.com', '600168163', '1986-07-27', 0, 26);    INSERT INTO borrowing (book\_id, reader\_id, borrowing\_date, return\_date, overdue\_penalty)  VALUES  (2, 2, '2022-09-02 11:43:03', '2022-09-13 10:16:19', 0),  (13, 2, '2022-09-02 11:43:36', '2022-09-13 18:30:26', 0),  (4, 1, '2022-09-13 18:31:18', '2022-09-21 13:01:06', 0),  (8, 6, '2022-09-27 12:20:25', '2022-10-11 15:50:31', 0),  (9, 8, '2022-10-02 17:00:42', '2022-10-18 11:09:12', 3),  (3, 2, '2022-10-07 09:37:26', '2022-10-20 11:00:43', 0),  (12, 2, '2022-10-07 09:37:40', '2022-10-20 11:00:59', 0),  (1, 4, '2022-10-30 16:11:56', '2022-11-21 12:40:41', 21),  (15, 7, '2022-12-02 15:50:42', '2022-12-19 15:15:16', 6),  (5, 3, '2022-12-11 18:47:01', '2022-12-30 12:01:24', 12),  (10, 2, '2022-12-11 10:13:43', '2022-12-25 13:30:20', 0),  (11, 2, '2022-12-11 10:14:04', '2022-12-25 13:30:33', 0),  (13, 8, '2022-12-13 14:04:30', '2022-12-28 11:00:33', 0),  # Insertions relative to present day for penalties and current borrowings to make sense:  (6, 15, DATE\_SUB(NOW(), INTERVAL 33 DAY), DATE\_SUB(NOW(), INTERVAL 7 DAY),  (((33 - 7) - (SELECT borrowing\_period\_days FROM library)) \* (SELECT late\_return\_penalty\_days FROM library))),  (7, 2, DATE\_SUB(NOW(), INTERVAL 19 DAY), NULL,  ((19 - (SELECT borrowing\_period\_days FROM library)) \* (SELECT late\_return\_penalty\_days FROM library))),  (12, 2, DATE\_SUB(NOW(), INTERVAL 17 DAY), NULL,  ((17 - (SELECT borrowing\_period\_days FROM library)) \* (SELECT late\_return\_penalty\_days FROM library))),  (15, 3, DATE\_SUB(NOW(), INTERVAL 16 DAY), NULL,  ((16 - (SELECT borrowing\_period\_days FROM library)) \* (SELECT late\_return\_penalty\_days FROM library))),  (5, 12, DATE\_SUB(NOW(), INTERVAL 18 DAY), CONCAT(CURDATE() - INTERVAL 1 DAY, ' 16:07:07'),  (((18 - 1) - (SELECT borrowing\_period\_days FROM library)) \* (SELECT late\_return\_penalty\_days FROM library))),  (6, 4, DATE\_SUB(NOW(), INTERVAL 10 DAY), NULL, 0),  (4, 4, DATE\_SUB(NOW(), INTERVAL 13 DAY), NULL, 0),  (11, 4, DATE\_SUB(NOW(), INTERVAL 13 DAY), NULL, 0),  (9, 12, DATE\_SUB(NOW(), INTERVAL 7 DAY), NULL, 0),  (2, 9, DATE\_SUB(NOW(), INTERVAL 4 DAY), NULL, 0),  (3, 1, CONCAT(CURRENT\_DATE(), ' 08:34:05'), NULL, 0); |



\* Nota: cabe señalar que el diseño de la biblioteca con las normas planteadas en este enunciado precisaría, para un funcionamiento congruente de su “business logic”, el uso de **procedimientos almacenados** que diariamente practicasen un incremento de 3 días de penalización para los usuarios que poseyeran libros retrasados en su devolución y un decremento de 1 día en la penalización de los usuarios que ya no tuvieran libros retrasados en su poder pero con una penalización en su cuenta > 0.

En este ejercicio de simulación, sin el uso de un sistema de procedimientos almacenados semejante, se producen algunas situaciones no deseables: por ejemplo, un usuario al que se fija una penalización por una devolución tardía, no importa el número de días de la pena, no podrá volver a tomar prestado un libro nunca al no practicarse la necesaria actualización diaria de penalizaciones mediante un procedimiento almacenado.